Programmazione procedurale

- È il tipo di programmazione su cui è basato ad esempio il C, usa il principio “divide et impera”: dividi un problema grosso in parti più semplici.
- L’attenzione è concentrata al problema. Questo viene risolto definendo **procedure** che interagiscono tra di loro.
- Applicazioni complesse vengono sviluppate spezzandole in componenti semplici per le quali risulta facile una risoluzione procedurale.

Programmazione procedurale

- Anche la programmazione procedurale consente **portabilità**: procedure esistenti possono essere riutilizzate.
- Nuove applicazioni possono essere create combinando in modi nuovi procedure esistenti.
- L'interazione tra procedure avviene tramite **condivisioni di dati** (passaggio di valori, passaggio di riferimenti, ritorno di variabili, variabili globali).

Programmazione procedurale

- In generale, lo scambio di dati può dare problemi di **protezione dei dati**.
- Aspetti connessi: scope delle variabili, definizioni di variabili come globali/locali, suddivisione dell'applicazione in moduli.
- Limitazione dell'accesso a certi dati: problema noto come **incapsulamento**. Spesso è un'informazione da inserire caso per caso.

OOP

- La programmazione a oggetti (Object Oriented Programming, OOP) invece concentra l'attenzione sui dati in gioco nel problema, e su come questi interagiscono tra di loro.
- L'applicazione nasce dall'**interazione tra oggetti**. Questo richiede:
  - che gli oggetti siano in grado di interagire
  - che la macchina capisca le loro interazioni
- Questa interazione genera le procedure.

Confronto

Procedurale

- Si progettano le procedure
- Si impone loro di scambiare dati

OOP

- Si progettano gli oggetti
- Si impone loro di interagire tramite procedure

OOP

- Nella OOP si usano oggetti la cui rappresentazione astratta (il paradigma degli oggetti) è detto **classe**.
- Nel definire una classe, il programmatore deve decidere di quali attributi (**dati**) e quali funzionalità (**metodi**) essa è dotata.
  - Tutti gli oggetti della stessa classe devono possedere questi dati e metodi (gli stessi)
  - Il programmatore deve anche decidere come gli oggetti interagiscono tra loro.

OOP

- In OOP, l'esecuzione di un programma consiste nella cooperazione di un insieme di oggetti.
  - La scrittura del programma corrisponde alla definizione di un insieme di classi (tutte quelle che occorrono per rappresentare gli oggetti in gioco)
- La OOP consente l'incapsulamento dei dati in modo molto più diretto ed efficace.
- Infatti, nella OOP dati e metodi sono intrinsecamente **specifici** di un oggetto e hanno significato solo in relazione ad esso.

La OOP consente quindi...

- **Maggiore semplicità rappresentativa**: basta descrivere con precisione gli oggetti che servono per l'applicazione.
- **Maggiore protezione dei dati**: le procedure sono metodi di un oggetto e hanno senso solo in relazione ad esso (non possono "sporcare" altri oggetti)

OOP e Procedurale a confronto

- La realtà di interesse di ogni problema è rappresentata dal calcolatore tramite il programma (che contiene solo gli aspetti ritenuti rilevanti dal programmatore).
- Nel caso procedurale, vengono descritte le procedure e le loro interazioni tramite dati.
- Nel caso OOP, vengono descritti gli oggetti e le loro interazioni tramite metodi.

OOP e Procedurale a confronto

- Esempio applicazione: acquisto di un vestito
- Descrizione:
  - un cliente entra in un negozio di abbigliamento.
  - chiede al commesso un vestito (di un certo tipo, con certi colore e taglia).
  - il commesso guarda in magazzino se è presente; in caso positivo comunica il prezzo; altrimenti l'applicazione si conclude senza acquisto.
  - se il cliente ritiene ragionevole il prezzo (confronta con budget) acquista l'abito; altrimenti l'applicazione si conclude senza acquisto.

Versione procedurale

```
Procedura main()
{ while (is_client == FALSE) loop;
  invoca req_cloth(type,size,col,budget);
  invoca price=check_price(type,size,col);
  if (price>budget) then exit(NO_SELL);
  else
  { invoca sell_cloth(type,size,col,price);
    exit(SELL);
  }
}
```

in rosso le procedure, in azzurro i dati

Spiegazione

- Una procedura **`main`** viene spezzata in più sottoprocedure:
  - **`loop`** gira finché non arriva un cliente.
  - **`req_cloth`** decide che vestito vuole il cliente.
  - **`check_price`** controlla se il vestito è in magazzino (trucco: ritorna **`+Inf`** se non c'è)
  - **`sell_cloth`** vende il vestito (e, ad esempio, decrementa la disponibilità in magazzino, aumenta l'incasso, etc.)

Versione a oggetti

```
classe Cliente
{ dati: type, size, col, budget;
  metodi: entra_negozio,genera_richiesta,
  acquista_vestito;}

classe Negozio
{ dati: stato,incasso,magazzino[];
  metodi: verifica_prezzo}
```

in rosso gli oggetti, in azzurro i metodi

Spiegazione

- Per ognuno degli oggetti, bisogna definire dati e metodi opportuni. Ad esempio:
  - il cliente ha un metodo **`entra`** per interagire col negozio; con questo metodo, lo **`stato`** del negozio passa da **`inattivo`** a **`in_servizio`**;
  - il negozio ha un metodo **`verifica_prezzo`**, che prende le richieste del cliente e le confronta con il dato **`magazzino`** (tabella di disponibilità e prezzi)
  - il cliente ha un metodo **`acquista_vestito`** che interagisce col negozio aumentando **`incasso`** e calando la disponibilità in **`magazzino`**

Versione a oggetti

- Bisogna ricordare che non sono gli oggetti, bensì le classi, a essere descritti direttamente.
- La classe viene definita descrivendo i dati e i metodi degli oggetti che vi appartengono (come un calco degli oggetti).
- Gli oggetti sono creati come istanze della classe. Oggetti della stessa classe hanno le stesse caratteristiche proprie della classe.

Versione a oggetti

- Uno specifico oggetto avrà un nome per referenziarlo. Ad esempio:
  - **`Dario`** e **`Luca`** sono oggetti della classe **`Cliente`**.
  - **`SartoriaChic`** e **`ModaDaPaolo`** sono oggetti della classe **`Negozio`**.
  - L'applicazione che comporta l'acquisto di un vestito è descritta in modo generale, sia che l'interazione avvenga ad es. tra **`Dario`** e **`SartoriaChic`** oppure tra **`Luca`** e **`ModaDaPaolo`**

Comportamento

- Gli oggetti sanno eseguire operazioni, in cui intervengono i loro dati e metodi. Tale complesso di operazioni possibili è detto **comportamento** dell'oggetto.
- Esempio del mondo reale: il televisore.
  - Operazioni che può eseguire:

    accensione, sintonia canale, variazione volume, spegnimento
  - Descrizione di alto livello (dall'esterno)

Messaggi

- Nel mondo reale, un'operazione che fa parte del comportamento viene attivata da un messaggio.
  - Ad esempio, il televisore viene azionato dal segnale inviatogli tramite il telecomando.
- Anche nella OOP, per fare eseguire a un oggetto un'operazione, è necessario inviargli un opportuno **messaggio**.

Interfaccia

- Per usare un oggetto, non è necessario sapere come è fatto dentro. Basta sapere qual è il suo comportamento e quali messaggi richiede.
  - Il manuale del televisore riporta il comportamento e quali tasti del telecomando vanno premuti.
  - Non riporta, ad es., lo schema del tubo catodico
- L'insieme delle informazioni necessarie all'utilizzo dell'oggetto è detto **interfaccia**:
  - operazioni che può compiere (comportamento)
  - formato del messaggio richiesto per eseguirle

Proprietà

- Gli oggetti possiedono a ogni istante di tempo un determinato **stato**, in cui sono assegnati certi valori a un dato numero di **proprietà**.
- Esempio del televisore:
  - accensione = valore booleano acceso-spento
  - volume = valore intero compreso tra 0 e *vol_max*
  - canale = valore intero tra 0 e 99
- Ogni proprietà possiede quindi un **nome**, un **valore corrente**, un insieme di **valori ammessi**.

Operazioni e proprietà

- L'effetto di un'operazione è quello di modificare lo stato dell'oggetto che la esegue, cioè cambiare il valore corrente di una o più proprietà (il nome della proprietà o l'insieme dei valori ammessi rimangono invariati!)
- Le caratteristiche di un'operazione, descritte dal suo **prototipo**, sono:
  - nome
  - parametri dell'operazione (e loro tipo)
  - tipo di ritorno

Programmare a oggetti

- L'OOP corrisponde a progettare oggetti (scrivendo classi). Tale progettazione è detta **implementazione**.
- Per un oggetto del mondo reale (es. televisore) la progettazione comprende due fasi:
  - definizione dell'interfaccia (decidere le operazioni che deve eseguire e i messaggi relativi)
  - realizzazione effettiva del progetto (che comprende anche dove assemblare il tubo catodico, lo schema elettrico da utilizzare, etc.)

Programmare a oggetti

- Anche l'implementazione di oggetti software prevede queste due fasi. In particolare la definizione dell'interfaccia.
  - è preliminare
  - è la sola parte visibile "dall'esterno"
  - richiede già di pensare alle interazioni che l'oggetto dovrà eseguire con altri oggetti: saranno possibili operazioni attivate da opportuni scambi di messaggi

Programmare a oggetti

- Per l'effettiva realizzazione del progetto, vale ancora l'analogia col mondo reale.
  - Un progettista di televisori non disegna su misura ogni componente elettronico, ma combina parti generiche, che trova sul mercato, e parti che disegna lui appositamente
  - Analogamente, il progetto di oggetti software fa largo uso di parti già esistenti.
- La buona OOP sfrutta la **modularità** degli oggetti, che ha come conseguenze importanti la **componibilità** e la **riusabilità**.

Oggetti software

- Un oggetto è un'entità software dotata di: **nome** che lo identifica, **stato**, **comportamento** (insieme di operazioni che può eseguire), **messaggi** che attivano le operazioni
  - comportamento + formato messaggi = interfaccia
- Le operazioni modificano lo stato dell'oggetto e consentono così le interazioni con altri oggetti.
- Le implementazioni delle operazioni sono dette **metodi**.

Oggetti software

- Gli oggetti che fanno parte di un programma OOP possono essere classificati come:
  - oggetti che modellano elementi concreti della realtà di interesse (esempio: pilota d'aereo)
  - oggetti che modellano elementi astratti della realtà di interesse (esempio: rotta dell'aereo)
  - oggetti introdotti per esigenze di implementazione (esempio: una finestra dell'interfaccia grafica). Questi sono oggetti virtuali che hanno senso solo nel contesto specifico del programma che li usa.

Esempio: `System.out`

- Ad esempio, Java può utilizzare un oggetto software pre-esistente chiamato **`System.out`**
- Questo oggetto rappresenta lo schermo del calcolatore. Dato che appartiene al package **`java.lang`** è automaticamente incluso durante la compilazione.
- Tra le operazioni che l'oggetto sa eseguire ce n'è una che produce la scritta di una frase sullo schermo. L'implementazione di tale operazione è il metodo chiamato **`println`**.

Esempio: `System.out`

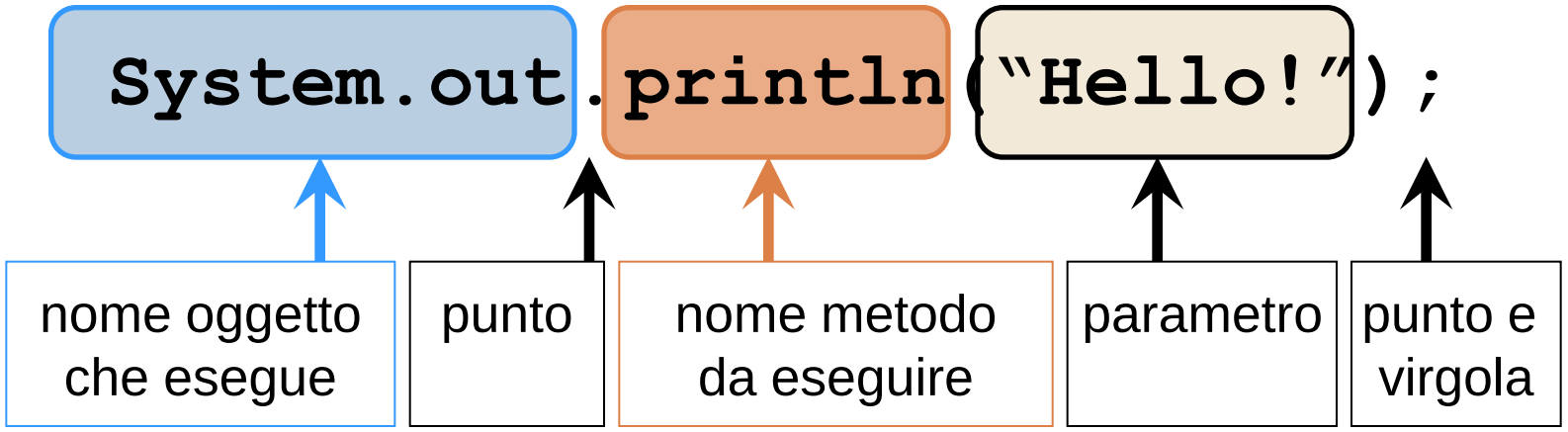
- Il messaggio da inviare all'oggetto **`System.out`** per fargli visualizzare la frase comprende il nome del metodo, **`println`**, e un parametro di tipo **`String`** che contiene la frase.
- Il prototipo di **`println`** è:

  `void println(String frase)`

  dove String è il tipo del parametro e void è il tipo di ritorno della funzione.

Esempio: `System.out`

- Per usare **`System.out`** per fargli eseguire il metodo **`println`** occorre perciò scrivere un messaggio in questo modo:



- Nota: quest'istruzione è solo un messaggio, non è un'applicazione eseguibile in Java.

Esempio: `System.out`

- L'oggetto **`System.out`** è un modello dello schermo nel senso che ne rappresenta alcune caratteristiche essenziali, ma non tutte.
- Il suo comportamento modella lo schermo solo come dispositivo per la visualizzazione di testi.
- Gli oggetti sono in grado di eseguire solo alcune operazioni ritenute significative nell'ambito della realtà di interesse.

Esempio: `Math`

- La classe **`Math`** serve come modello di calcolatore, nel senso che sa svolgere operazioni per il calcolo di funzioni elaborate:

  `double sqrt(double x)` per: $\sqrt{x}$

  `double pow(double b, double x)` per: b^x

- Di solito i linguaggi di programmazione forniscono solo operatori elementari: + - * / etc. oppure usano librerie (da inserire in blocco).
- In Java la classe **`Math`** è fornita con **`java.lang`**

Creazione di oggetti

- Un oggetto è un'istanza di una determinata classe. Per crearlo e usarlo bisogna:
  - dichiarare una variabile per l'oggetto
  - creare l'oggetto memorizzandone un riferimento nella variabile corrispondente
- A questo punto l'oggetto può essere usato inviandogli opportuni messaggi.
- L'oggetto va referenziato usando la variabile corrispondente.

Classi

- Per programmare in Java è necessario scrivere classi (programma = insieme di classi)
- Una classe deve descrivere in astratto gli oggetti che ne saranno istanze, dal punto di vista di stato e comportamento:
  - lo stato è un insieme di proprietà
  - il comportamento è un insieme di operazioni

Classi

- La classe viene descritta attraverso:
  - i dati (che rappresentano proprietà)
  - i metodi (che rappresentano operazioni)
  - inoltre bisogna anche descrivere le azioni da svolgere quando viene costruito un nuovo oggetto di quella classe.

Metodi

- Un’istruzione è il singolo mattone costitutivo di un metodo. I metodi sono gruppi di istruzioni, riuniti per ottenere una singola funzionalità (ovvero, implementare un’operazione).
- In effetti, sono abbastanza simili nella sintassi alle funzioni di ANSI C.
- Tuttavia, sono specificati relativamente e internamente a **un singolo oggetto**.

Metodi

- La sintassi è:

```
tipo_ritorno nome_metodo (tipo_arg arg, [...])
{
    //istruzioni
}
```

- **`tipo_ritorno`** e **`tipo_arg`** sono tipi di dato
  - come vedremo, questi tipi possono essere primitivi o riferimento
  - se il metodo non ritorna valori, **`tipo_ritorno`** va specificato come **`void`**.

Metodi

- Le variabili definite in un metodo sono locali, visibli solo all'interno del metodo stesso:
  - il ciclo di vita è limitato all'esecuzione del metodo
  - non sono accessibili da altri metodi
- Per evitare l'allocazione di variabili inutilizzate, il compilatore prevede un controllo
  - un metodo non può essere compilato se contiene variabili a cui non viene assegnato alcun valore
  - in tal caso la compilazione si interrompe (errore)